Ipipri Digital Trails: Augmented Reality Experience

Liam Fenneman

***Abstract*—Augmented reality is a growing technology about bridging the gap between the virtual world and the real world. Most people carry around a mobile phone that is equipped with a rear-facing camera and a touch screen. Augmented reality applications make use of the rear-facing camera to place virtual elements such as three-dimensional models into the physical environment. The Ipipri digital trails project is a larger project developed by the Russell Museum that aims to showcase Māori history in Kororāreka (now known as Russell). This project aims to develop augmented reality elements that will enhance the experience of users by showcasing three-dimensional models that people can view anywhere. Augmented reality is a relatively new technology with many features being experimental which comes with large trade-offs. The biggest of these trade-offs is speed versus accuracy. This project leans towards the speed side of this trade-off as the accurate placement of models was not a priority. The solution that I have developed uses persistent ray casting to determine where in the physical environment should be the origin of the virtual scene. This allows for the instant placement of models into the virtual scene and then uses depth information to further refine the virtual scene. Along with the augmented reality elements my solution also includes a quick-response (QR) code scanning system. This allows users to scan a code to choose which model is used for augmented reality. This project is important to help make learning the history and stories of Kororāreka more engaging by giving people a visual element that is hard to otherwise recreate.**

I. Introduction

Augmented reality (AR) is a growing technology that aims to bridge the gap between the real world and the virtual world. This is done by using a mobile device's camera as the view into the real world and then placing virtual elements in front of the real world. This allows for the perception that those virtual elements are in the real world. The key issue that has made AR difficult to develop is how to map a virtual scene onto the camera view in a way that makes the virtual scene look like it is part of the real world.

Kororāreka (now known as Russell) is a small town in the Bay of Islands, New Zealand. It has a long and rich history for both Māori and Europeans. Before the arrival of Europeans, Kororāreka was a Māori settlement. When visiting Russell many people focus on European history which can overshadow the rich Māori history and stories. The Ipipiri Digital Trails project by the Russell Museum is a project that aims to showcase the Māori history of Russell.

Attracting tourists to Russell is important for the Russell Museum as well as other businesses in Russell since tourism is a major component driving the local economy. By creating an AR application the Russell Museum can enhance the experience of tourists and potentially entice more tourists to come to Russell. This theory is supported by a 2022 study that showed innovation and AR quality are important factors in influencing a person's attitude to AR [1]. This can then lead to a positive impact on the intention to visit a particular destination [1].

This project was supervised by Kevin Shedlock.

Fig. 1: Example of how the map and QR codes could look on a pamphlet.

The primary problem being solved in this project is the AR experience component of the Ipipiri Digital Trails project. The Russell Museum is developing a pamphlet that will be sold which will be the primary way that people will be able to access the AR experiences.

The pamphlet includes a fold-out map of Russell with QR codes spread along the edges similar to what is shown in Figure 1. The QR codes are needed to be able to view any of the AR experiences. For this project, I implemented the scanning of the QR codes and the mapping of the QR codes to various AR experiences.

The solution that I have developed is an Android application that is built using the Unity game engine. This application handles both the QR code scanning and the creation of AR experiences.

The system for viewing the AR experiences allows users to place a three-dimensional (3D) model into the real-world environment. In total, nine AR experiences can be viewed. Each experience is associated with a QR code. The AR experiences are Frigate L'Aurore, hakari, kiwi, kororā (little blue penguin), whale tooth scrimshaw, a shark jaw, beetle whaleboat, giant weta, and a small Māori village scene. The models for this project were provided by the Russell Museum. Figure 2 shows an example of a model that has been placed in the real-world environment.

II. Related Work

Augmented reality is a technology with multiple different methods of creating an experience. This is best described in a 1995 paper that discussed the existence of a "Reality–Virtuality continuum" where both AR and virtual reality (VR)

Fig. 2: The penguin model placed in the real-world environment.

existon a spectrum with multiple levels based on the particular implementation [2]. In this paper, Milgram et al. described two types of displays for AR: see-through and monitor-based [2].

See-through displays work by embedding a display into a see-through panel usually made of glass. This can be used to eliminate the perspective problem that occurs with monitor-based displays, however, this approach comes with its problems. The first is the balance that has to be achieved between displaying a clear graphic and blocking the vision of the user. If the graphics completely block the user from seeing through the panel then it just becomes a regular display, however, if the graphics are barely visible then the experience can be very poor.

An example of a see-through display is found in the helmets of F-35 pilots [3]. A heads-up display (HUD) is mounted to the pilot's helmet which provides them with information from the aircraft and its sensors. While the HUD is intended to be the "primary display system" for the pilot, it also allows the pilot to retain vision within the cockpit [3].

This use case is not only applicable to fighter pilots but also to regular people in the form of a HUD for their car. Figure 3 shows how this HUD can work. This particular implementation also includes other screens such as Global Positioning System (GPS) navigation and phone notifications [4]. This example shows how the balancing problem can be solved by reducing the opacity of user interface (UI) elements and keeping the size of the panel large enough to see the content but small enough to not block the whole vision of the user.

Monitor-based means that the environment is recorded and digitally overlaid onto a monitor [2]. Since this paper's publication in 1995, smartphones have become a piece of technology nearly everybody carries around with them. This is important because a smartphone contains a digital camera capable of recording the environment, a high-quality display, and a processor capable of taking a live recording of the environment and mapping it onto the built-in display. This is a major advantage of this type of approach as it is the most accessible since most people carry a smartphone with them and, therefore, don't require any additional hardware to be able to use AR.

An example of the monitor-based approach is the NosferARtu project which was built to "explore the possibilities and advantages of the gamification methods with the combination in Augmented Reality technology" [5]. This application takes users around the Orava Castle in Slovakia collecting virtual objects that are placed around the castle. While the project was done to explore the impact of the gaming element, it also highlights the advantages and disadvantages of the monitor-based approach to AR. As shown in Figure 4, the phone camera takes up the entire screen and a virtual object (the statue) is overlaid to give the perception that the statue exists in the real world.

A key disadvantage of the monitor-based approach is that the perspective that is being displayed is not the user's perspective but rather the camera's perspective. For an application like NosferARtu, this might not degrade the experience, but in other use cases for AR this might be a factor to consider.

A 2016 study was done to find the user requirements for wearable AR glasses [6]. The study was done in the context of an art gallery and made use of a see-through display to show the user information about the art piece they were looking at.

When the user looked at a piece of art they would be shown additional information about the painting in the form of text [6]. One of the observations that was made from this study is the requirement that the information needs to balance along the line of providing enough usefulness while not providing too much information.

Another important observation that was made is the novelty factor which is a "crucial determinant of participants' perception" of the usefulness of the AR application [6]. What this could mean is that as AR becomes more popular the novelty factor will decrease and therefore could affect the perception of the usefulness of AR applications in terms of attracting

Fig. 3: Example shot of the HUDWAY Drive panel. Source: [4]

Fig. 4: Monitor-based AR in the NosferARtu project. Source: [5]

tourists.

One of the reasons why this project is being done is to explore the use of AR for tourism activities. One study that has been done looking at this exact topic is a 2022 study that looked at the use of AR as a way for people to have a positive experience before they decide to travel to the destination the AR experience is made for [1].

In this study, the researchers aimed to find the motivating factors of AR that would positively influence a person's attitude towards the destination [1]. They conducted a survey and found that the quality of the augmentation and the perceived usefulness of an AR application are factors to improve a person's attitude towards AR which contributes to a person's expected enjoyment of the destination [1]. The expected enjoyment then leads to a positive impact on the intention to visit the destination.

A major limitation of this study is the fact that it was only proven by the results of a survey. What people say they might want is not necessarily how they would actually behave when given an AR application.

This study also only considered the impact when the person uses the AR application before travelling to the destination. This means that for this project the outcomes could be different since the people who will use the application must first be in Russell to get the pamphlet.

III. Design

A. Constraints

One of the key constraints on the project is the fact that I am the sole developer. This means that my time spent working on the project is limited. The scope of the project must also be limited to take into consideration this fact.

Another constraint for this project is the lack of mature frameworks, guides, articles, and research for augmented reality. There is no clear pathway for creating AR applications and therefore many features are experimental and there are no industry standards to provide compatibility between different tools.

In terms of object tracking systems, many require some form of physical thing to exist in the real world to generate the virtual world from a known point. However, for this project, the user is only expected to need the pamphlet and no other physical items. This limits the choice of object tracking as it will have to be done purely using the users' camera.

B. Scope

As iteration occurred the scope of the project along with the requirements changed to make the project not have to wait for outside parties. An example of this is the change from using a virtual map that uses AR to a QR code-based solution. This decision was made since this would require working with a digital artist to create the map and get it integrated with the application I built. When taking into consideration the time constraint this decision is sound.

The primary deliverable for this project is a working application with AR elements. For this, we agreed that 10-12 models would be used to create AR experiences. However, only 9 models were provided and therefore only 9 AR experiences are included in the application.

Also included in this project is the QR system which scans QR codes and uses the text that is contained in the codes to determine which AR element to show to the user.

The application also needs to include a basic UI to move between different states of the application. The UI includes a slider to control the rotation of the AR element. The design of the UI is not a part of the scope of this project. However, it should include basic usability requirements.

To keep the scope focused on the AR technology and implementation I decided to only build the application to target Android devices. Building an AR application for iOS would require setting up and testing both Android and iOS devices which each have their corresponding software development kit (SDK) for implementing AR.

While all of the tools that I have used for this project have integrations and systems to support iOS development there is still the issue that iOS development must be done using a machine running MacOS. To test that the application works on iOS I would also need access to a phone that is running iOS.

Building for Android only makes managing the build system significantly easier since I only have one target. It is also made easier by the fact that I have all the equipment to be able to run and test the application.

C. Methodology

The process for designing this project was iterative as I explored what is possible within Unity/ARCore and adjusted based on feedback given from both Kevin and the Russell Museum. This was done to find a solution that best fit the needs of the Russell Museum.

Within the iterative process, I designed around creating a single working AR experience which can then be duplicated and scaled to support multiple experiences. This was to limit

the complexity of the AR elements and allow for any number of AR elements to be included in the final artefact.

The problem that is being solved in this project is split into two systems. The first is the QR code system and the second is the AR elements and experience system. These systems can be decomposed since they are mutually exclusive in terms of when they are used.

The AR system has to wait until the user has chosen an experience before any work can be done to display AR to the user. The AR system doesn't need to care about how the user selects an experience since the system is only responsible for handling the display of AR elements.

This contrasts with the QR code system which is only responsible for scanning QR codes and telling the AR system which experience the user has selected. This is done to minimise the dependency boundaries between the two systems.

Creating the mapping between the virtual world and the physical world is one of the most important decisions to make as there are many trade-offs to consider. With ARCore there are four different methods to create this mapping: anchors, planes, raycasts, and tracked images [7]–[10]. Table I shows a simplified comparison of the trade-offs between each method. Each option has its advantages and disadvantages which makes it more or less useful for different applications.

The purpose of AR in this project is to place a 3D object at any point in the environment. The two types of trackable objects that are viable for this project are plane detection and raycasts because they don't require any physical thing to create the virtual scene. The trade-off between the two can be reduced to a simple factor: ease of setup and speed to initialise.

D. Functional Requirements

As shown in Figure 5, there are two core systems for the application: the QR code system and the *AR Experience* system.

As the name suggests, the QR code system has a single responsibility: scan for QR codes. Therefore, the first requirement is that the system can scan text-based QR codes.

The QR are going to be placed on a pamphlet which means that the size of the QR codes is small. The second requirement for the QR code system is that the system must be able to handle physically small QR codes. However, this issue is partially solved by how the QR codes are generated in the first place. Figure 6 shows a comparison of two types of QR codes that are possible to create. Figure 6a shows a QR code generated by a short text and Figure 6b shows a QR code generated by a long text. The difference is clear that the longer the text the larger the QR code will be. By using the short text

TABLE I: Comparison of AR tracking methods. Source: [7]–[10].

Method	Advantages	Disadvantages
Anchor	Good accuracy	Difficult, Resource Intensive
Plane	Easy to set up	Slow, Average accuracy
Raycast	Fast to initialise	Poor accuracy
Tracked Image	Good accuracy	Requires image to be present in real-world

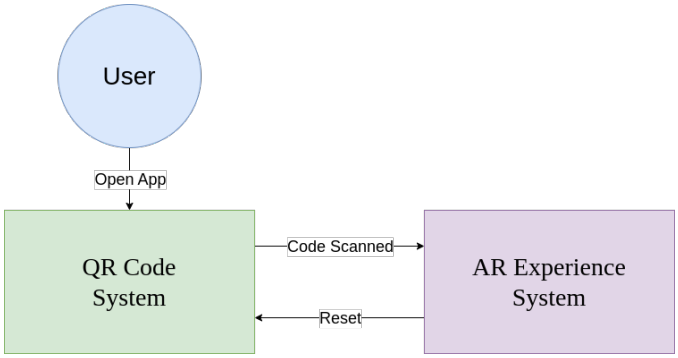

Fig. 5: System flow diagram of the application.

method the QR codes can be made small enough to fit on a page of a pamphlet while still being large enough to be easily scanned.

The third requirement for the QR code system is the user needs to be given feedback from the system about what it is doing and what has been scanned. Multiple QR codes will be on the same page of the pamphlet which means that the user needs to know what they have scanned before the AR experience starts. This is as simple as displaying a message to the user saying "scanning" when the system is still scanning and the name of the AR experience when it has successfully scanned.

The input to the QR code system are frames from the device camera. Since the camera is used for the background of the application, the QR code system can hook into the Unity renderer with a callback when frames are created. Within this callback, the QR code system can scan individual frames from the camera.

Upon successful scanning of a QR code, the QR code system will update a globally readable string with the content of the QR code and pause. The QR code system will only resume scanning upon receiving the "Reset" signal.

The AR experience system has the responsibility of displaying the AR elements to the user. The first requirement for this system is that it needs to allow the user to place an object anywhere in the environment. This means that the user doesn't need to be at a specific location to be able to use the application.

The second requirement for the AR experience system is that it needs to support multiple AR experiences. While an

(a) QR code with the text: "Kiwi."

(b) QR code with the text: "This is a long QR code."

Fig. 6: Example of version 1 and version 2 QR codes.

application with a single experience might be interesting, the ability to support any number of experiences is important for future development. This also allows the users to be able to decide what they want to view.

The third requirement for the AR experience system is that it needs to implement some form of interactivity. By using AR you are not limited in the level of interaction that you can provide to the user as you would be with a video or picture. By taking advantage of the game engine nature of developing an AR application we can implement some of the same ideas that are done when developing a game.

E. Performance Requirements

The project's dependence on ARCore means that the minimum version for Android is set by ARCore which is Application Programming Interface (API) level 24 [11]. This is a fairly old version which was released in 2016 and has a cumulative usage of 96.19% [12].

The biggest performance requirement for this project is the target frames per second (FPS) which determines how smooth the experience is for users. As this project is a real-time application this is important to keep as high as possible. For this project, I target 60 FPS which is a standard target for real-time applications. This target means that there is less than 16.67 milliseconds to complete any work as blocking the main thread for longer would lead to missing the target.

An alternative FPS target is 30 FPS which is the default for mobile applications in Unity. The time between frames at this target is double that of 60 FPS, however, I decided to use a higher target until I needed to drop that requirement. This is because the two primary systems can both be designed to complete work over multiple frames.

The QR code system's primary performance requirement is that it needs to provide quick feedback to the user. Ideally, this feedback is in the form of a successful scan. However, the user must be made aware of what the system is doing. This is important because the user could believe that they are doing something wrong when they have done everything correctly but the system is processing a QR code. This could lead to a frustrating situation for the user. A reasonable window for scanning a QR code would be within 1 second assuming that the QR code is within view of the camera.

Similarly, the AR elements need to be placed quickly. This is for the same reasons as the QR code system since the user needs to be given feedback that the AR elements are being placed. A reasonable window for placing the elements would be within 1 second of the user tapping the screen.

Another requirement for the AR elements is that they need to be placed reasonably accurately. AR technology is limited in the accuracy that it can achieve, however, the accuracy needs to be good enough that the user can easily see the AR element in a realistic position. This is further limited by the constraint on supporting hardware that would allow for more advanced tracking such as the use of anchors.

F. Usability Requirements

The target audience for this project is mostly tourists. This poses some challenges when it comes to designing the usability of the application. Firstly, there are likely to be users who have limited or no understanding of English. Secondly, there is no targeted age range which means that users of all ages could be using the application and therefore consideration needs to be made to ensure that the UI is accessible for all ages.

To be able to objectively consider the usability requirements of the project I used Nielsen's heuristics to guide features that need to be implemented to meet the usability requirements that users expect. [13].

The first of these heuristics requires that the application is designed to "always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time" [13]. This was partially discussed in the performance requirements section, however, it is also important to consider the UI elements that are used to provide feedback to the user.

The second and fourth heuristics are concerned with the language used within the application. The second heuristic states that the application should "speak the users' language" and the fourth heuristic states that the application should "maintain consistency within a single product" [13]. What this means for this project is that any text used is widely used by the target audience and remains consistent across the entire application.

The third heuristic is concerned with the user's ability to move freely between different states of the application. For this project, this is reasonably simple as there are only two systems that the user can interact with. Therefore, the user should be able to move between the QR code system and the AR experience system. This can be achieved by using a button to trigger the transition.

The remaining heuristics are more applicable to larger applications that have more complex UI elements and user interactions. However, some important takeaways from these heuristics are that the application should be designed to keep users in a valid application state, and help users recover from errors. The design of the application should only provide a minimal UI to reduce the cognitive load on the user.

IV. Implementation

A. Dependencies, Tools, and Frameworks

The first step in the process of implementing this project was to decide on the tools that I would use to build the application. The problem this project is solving is not the process of building an application but rather the systems required to create AR elements. Therefore, I decided to use a game engine rather than building a native application as this would allow me to focus on the creation of the systems required to create AR experiences. The following list of features that are provided by a game engine are systems I would have to create if I decided to build a native application:

- Build system to generate Android package (APK) files to deploy to Android devices.
- Dependency management.
- Debugging tools such as logging and a profiler.
- Rendering pipeline for 3D graphics.
- Asset management.

- Mobile input handling.
- UI development.

The prototype I built before my trip to Russell was using the Unreal game engine. However, this was changed to Unity as I found that the version of ARCore that was being used was older than the version that Unity was using. This meant that some features that I wanted to use such as the *Depth API* were not available.

Unity's package management system handles the dependencies for the project. This means that I just need to define which dependencies I want to use and they will be included when I build the project. This includes the Android SDK, ARCore, and the ZXing library.

One of the advantages of using Unity as the build system for this project was the fact that I didn't have to directly interface with the Android SDK. However, I made use of one of the many tools that come with the Android SDK. *Android Debug Bridge* is a tool that allows me to connect to my Android device over the Universal Serial Bus (USB) interface or wirelessly and run my application [14]. Since Unity also has integrations for Android that make use of these tools I was able to make use of incremental compilation which significantly reduced the time it took to build and deploy the application to my device.

ARCore is the most important dependency for this project as it provides the API to be able to create AR experiences. However, I use ARCore through the use of a Unity package called *AR Foundation* which is a cross-platform API that exposes features of the ARCore API [15]. This is done since *AR Foundation* also supports iOS, among other platforms, which means that the same code can be used to build for multiple platforms in the future. *AR Foundation* itself doesn't provide any functionality but instead maps the APIs of different libraries to a single API that can be used within C# [15].

To handle the scanning of QR codes I decided to use a third-party library called *ZXing.Net* [16]. This library provides support for scanning QR codes and is compatible with Unity [16]. The reason that I am using a library rather than implementing the scanning myself is that the *ZXing.Net* library has been developed over many years and includes many optimisations that I would not be able to implement in the time frame of this project. *ZXing.Net* is provided as a dynamic-link library (DLL) which means that it can be included in the project as a plugin and used within C#. Unity will handle the management of the DLL and include it when the application is built.

B. Scene Management

Unity makes use of an abstraction called *scenes* to manage what assets and scripts are included in different parts of the application. This abstraction allows for multiple scenes to be loaded at different times based on what the user is doing. For this project, I made use of two scenes.

The first scene is the entry point to the rest of the application. Within this scene is the UI that the user first encounters which contains the title of the project and a start button. When the start button is pressed the second scene is loaded and the first scene is unloaded.

The second scene is the primary scene that contains both the QR code system and the AR experience system. The reason why this scene contains both the QR code system and the AR experience system is because they both make use of the phone camera. Splitting this scene in two would require the camera to be initialised twice which would lead to a moment of blackness on the screen which switching between the two.

The entry point scene is split from the main scene since this allows for multiple development scenes to be loaded at the same time to test different configurations of the QR code system or AR experience system. For example, when developing the QR code system I didn't need any of the AR elements to be loaded which meant that they didn't interfere with the development of the QR code system. I was also able to build an application that could use either plane detection or persistent raycasts by using a different scene for each configuration. This allowed for testing the different configurations without having to change any code and rebuild the application.

C. AR Experience System

The AR experience system is responsible for displaying the AR elements to the user. While there are multiple ways of implementing the system, I decided to use a finite state machine which allows for the representation of the entire system to be made explicit. This is done by defining the states that the system can be in and the transitions between those states.

The benefit of using a finite state machine is that it allows for asynchronous behaviour to be created. This is important for this project as the performance requirement of targeting 60 FPS only allows for 16.67 milliseconds to complete all work before the next frame must be rendered. Allowing for the work to be done asynchronously means that the work can be split up over multiple frames.

One of the challenges with an asynchronous system is that it can introduce bugs that are difficult to reproduce and debug. However, using a finite state machine to control which states the system is allowed to be in will ensure that the system is not able to be in an invalid state.

The AR experience system is implemented as a Unity game object called the AR Experience Manager. For every frame Unity will call the Update function within this system which will then do whatever is needed based on the current state of the system. This allows for creating an asynchronous system since the Update function can return and wait until the next frame.

As shown in Figure 7 the AR experience system contains three states: *QR*, *Placement*, and *AR*.

When the AR experience system is in the *QR* state it will check for updates from the QR code system and ensure that the QR code system is in the correct state to scan for QR codes. This is to ensure that the QR code system is only scanning for QR codes while the user is expected to be scanning for QR codes. When the QR code system has successfully scanned a QR code the AR experience system will check to see if the text that the QR code contains matches any of the AR

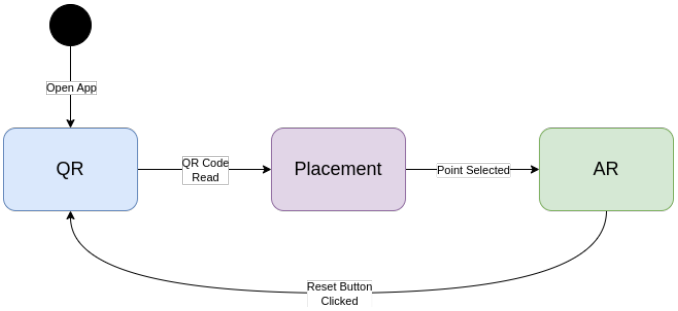

Fig. 7: State transition diagram of the AR experience system.

elements that the AR experience system is storing. If there is a successful match then the state of the AR experience system is transitioned to the *Placement* state. However, if the text doesn't match any AR elements then the AR experience system stays in the QR state.

The *Placement* state exists to allow for an in-between state to exist that allows the user to select where they want to place the AR element in the environment. This state also allowed me to display additional information to the user during this time. A secondary advantage of using this in-between state is that it allows the user to scan a QR code and physically move their phone to a different location before the AR element is placed. This is important for the user experience since once the object is placed the location is not able to be changed.

To transition to the *AR* state the user must touch their screen to place the selected AR element. This is done by using Unity's Input system which provides a screen-space coordinate of where the user clicked. When the AR experience system detects the user has touched their screen a persistent raycast is created using the AR Foundation API. This raycast is then used to instantiate a new instance of the AR element that the user had selected and the state of the AR experience system is transitioned to the *AR* state.

The *AR* state doesn't have much functionality other than changing the UI to display a slider and hide the help panel. This is because the ARRaycastManager that is provided by *AR Foundation* will ensure that the mapping between the virtual world and the real-world is updated as more information from the phone's camera and depth sensors (if available) is collected. This means that once the AR element is placed in the virtual world the position within the virtual world doesn't need to be updated.

D. Plane Detection vs. Persistent Raycasts

Both plane detection and persistent raycasts are methods that can be used to generate the mapping between the virtual world and the real world. They both make use of the ARRaycastManager that is provided by *AR Foundation* which then uses *AR Core* to do the work of generating the mapping.

The initial implementation of this project made use of plane detection since this is the easiest method to get started with. Figure 8 shows the steps in the process of placing an AR element using plane detection. The scanning for planes is done by the ARRaycastManager which will take the most amount of time as the user needs to move their phone around to gather as much data as possible which can be used to generate the mapping. Once a plane is detected an ARPlane object is created which can then be used to place other objects on top of it.

The process of detecting planes has a major disadvantage since the time it takes to detect a plane can be as long as 20 seconds. This is not acceptable to achieve the performance requirements of this project.

Persistent raycasts are similar to plane detection except that instead of waiting for a full plane to be created the system will create a raycast from a certain point on the screen and estimate the distance from the camera to the surface that the raycast intersects. The estimated distance is something that I can directly control and have set to 3.5 meters. Figure 9 is a bit more involved since after the raycast is added to the ARRaycastManager the flow of the system splits to allow for an AR element to be immediately placed at the intersection point while the ARRaycastManager continues to update the raycast.

The reason these raycasts are called persistent is because internally they are constantly being updated to ensure that the point of the intersection remains the same. This means that as the user moves their phone around the raycast will still intersect with the same point in the real world. By constantly updating the point of intersection the accuracy can also be improved as more information is collected.

I was able to do some basic manual testing of both methods and found that plane detection was taking around 5 seconds from starting the application to having an AR element placed. Persistent raycasts, on the other hand, were able to reduce this to under 1 second. This is a significant improvement and therefore I kept the persistent raycast implementation.

The difference between the two implementations in terms of accuracy is negligible as long as the raycast point remains in roughly the same spot when using the persistent raycast implementation. This is because the initial accuracy of persistent raycasts cannot be as good as the accuracy of plane detection since accuracy improves over time. However, both implementations suffer from accuracy issues when the user is moving too much or too quickly so this is a minor issue.

E. QR Code System

Similar to the AR experience system the QR code system also makes use of a state machine to decide what the system needs to be doing. The importance of using a state machine is greater with this system since blocking the main thread would mean that the camera is not being updated which would effectively freeze the application.

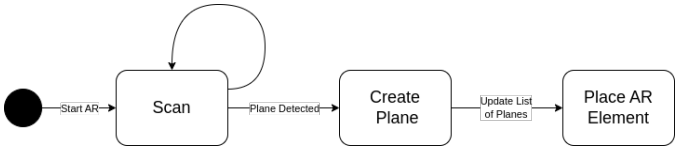

Fig. 8: Plane detection implementation diagram.

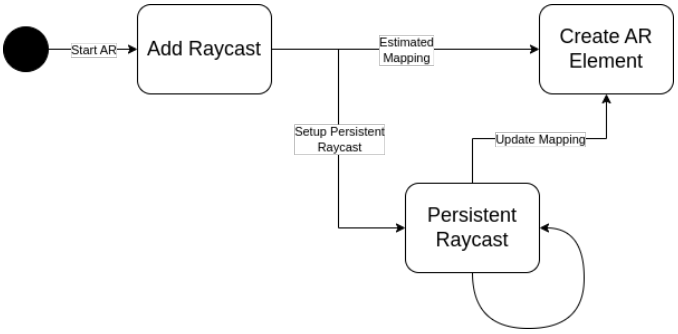

Fig. 9: Persistent raycast implementation diagram.

Figure 10 shows the state machine diagram for the QR code system which has three states: *Idle*, *Scanning*, and *Done*.

The QR code system starts in the *Idle* state since this allows for other systems in the application to initialise and then request that the QR code system start explicitly. This is because there is no point in handling any errors that could occur from starting the QR code system if the AR experience system hasn't been initialised and waiting for a QR code.

The AR experience system must explicitly call the Scan function to transition the QR code system into the *Scanning* state. When a QR code has successfully been scanned the state transitions to the *Done* state.

The scanning is done by listening to the OnCameraFrameReceived event that is called after the current frame is rendered. This event makes use of the rendering pipeline for the ARCamera which is responsible for rendering the background of the application. This means that the QR code system can hook into the rendering pipeline and scan the latest frame without needing to initialise its own camera feed.

This event listener checks for the current state of the QR code system. If the state is *Scanning* then the current frame is scanned for QR codes. Otherwise, the event listener will return and wait for the next frame to be received. This means that the *Idle* and *Done* states are functionally identical. However, *Done* indicates to any system that the value currently held by the QR code system is the latest valid QR code that has been scanned. The *Idle* state only indicates that the system is not scanning for QR codes and the QR code result stored by the system could be old or invalid.

The implementation of scanning a single frame is done by an external dependency, *ZXing.Net*. This is because the implementation of scanning a single frame is complex and requires a lot of optimisations to be able to accurately and quickly find QR codes. This would be a significant amount of work to implement myself and is ultimately out of the scope of this project.

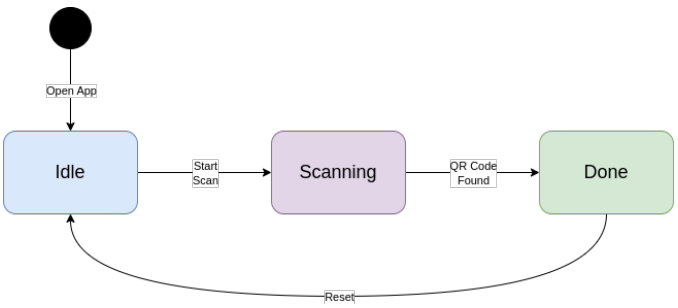

Fig. 10: State machine diagram for the QR Code system.

F. Scriptable Objects

One of the features that Unity provides is called *scriptable objects* which are objects that can be created within the Unity editor to store data for use within a C# script.

I make use of *scriptable objects* for storing information the name, text used for the QR code, and the 3D model for each AR experience. These objects are then added to a list within the *Experience Manager*.

Since *scriptable objects* are a type of Unity asset like a texture or 3D model they can be checked out with *git* which means that any changes are tracked and can be reverted if needed.

Another advantage of using *scriptable objects* is the fact that creating them and adding them to the *Experience Manager* can be done entirely within the Unity editor and without any changes to the code.

G. User Interface

The user interface is implemented using Unity's built-in UI development system which makes use of pre-built game objects for various UI elements.

In terms of the typography that is used throughout the application, I decided to import the same font that is used on the Russell Museum website. This is because I wanted to make the design feel relatively consistent with the Russell Museum brand without having to follow an explicit design guide.

The UI for the entry-point scene consists of a title and a start button. I wanted to ensure that the UI is as minimal as possible to reduce the options that the user has to think about to get to the AR experiences.

Within the main scene, the UI remains minimal by providing only the information that the user needs to be able to work the application. Four UI elements are used within the main scene: a text element at the top of the screen, a help panel, a slider, and a reset button.

The text at the top of the screen shows the name of the AR experience when the user has scanned a QR code or a message that indicates that the system is expecting the user to scan a QR code.

The help panel is a panel at the bottom of the screen that shows some helpful information about what exactly is expected of the user a various points of the application. For example, when scanning for a QR code this panel will display: "Place a QR code in the center of your screen." When the user has scanned a QR code this help message is changed to: "Tap to place the object." This panel was implemented to ensure that the user is not confused about what they need to do at any point in the application.

When the user has placed an AR element a slider will appear at the bottom of the screen which allows the user to change the

rotation of the AR element. This is because physically moving around the element is not always possible and having a slider is also more convenient.

A reset button is always present in the bottom right corner of the user's screen that will allow the user to reset to the *QR* state and will ensure that the QR code system is also reset. This allows the user to freely move between different AR experiences without having to restart the application.

V. Evaluation

The functional requirements for the QR code system required that the user can scan QR codes, and handle when the QR codes are small. This was achieved by using a third-party library that has implemented many features that make the scanning of small QR codes possible. The minimum size that a QR code can be is 2 centimeters as measured from the perspective of the phone camera. This is something that could be different on different devices since this is affected by the camera's ability to focus on code well enough to be able to scan it.

The performance of the QR code scanning system is as good as it can be since the time to scan a QR code is less than 100 milliseconds. This is well within the performance requirement of less than 1 second. The feedback from the QR code system is done at the same time which means that the user can immediately see if the QR code that they have scanned is valid.

For the AR experience system, one of the functional requirements was to allow the user to place an element anywhere they want to. This was achieved by using persistent raycasts which converts the screen-space point from where the user tapped their phone into a world-space point which is used to place the AR element. This is further extended by using the *Placement* state in the AR experience system which allows users to scan a QR code and physically move before tapping their screen to place an element.

Supporting multiple experiences was one of the most important requirements for this project and was achieved by using a list of *scriptable objects* that contain all the information that the application needs to be able to place arbitrary 3D models in the environment. In total, I was provided with 9 3D models which all have a unique QR code that is used to identify them.

The interactivity is done by using a slider that allows the user to rotate the AR element. This is a simple implementation, however, the user also has the ability to physically move around to view the AR element from different angles.

To achieve the 60 FPS performance target I ensured that all of the systems were designed to be able to complete their work over multiple frames. This was done by implementing a basic async runtime with the use of state machines. This metric is hard to evaluate after reaching the point of 60 FPS since allowing the application to produce more than 60 FPS has little effect on the performance of the application since most phones' refresh rate is 60 Hz so frames are being created and dropped without the user ever seeing them. However, producing more frames could impact the battery life of the device since the phone is creating frames rather than being idle.

The placement of AR elements is also within the 1 second window outlined in the performance requirements. This was done by using persistent raycasts which allow the placement of an AR element to be done as quickly as possible by removing the need to wait for plane detection. This comes at the cost of the immediate accuracy since there is not as much information being used to generate the mapping between the virtual world and the real-world. However, the accuracy improves over time as more information is collected by the phone's camera and depth sensor.

VI. Conclusions & Future Work

This project is a great place to start implementing a more featureful application that could be used by the Russell Museum. The project has achieved the primary goal of creating a system that allows for the placement of AR elements, however, to make a successful application many more features could be added to make the experience of using the application better.

One of the first features that I would implement is building the storytelling elements. The current state of the project can show users 3D models of things that they might've never seen or be able to see up close, however, there is no contextual information about the models. For example, the hakari model was designed from a painting of a hakari that was once located in Russell.

Similarly, the project could be extended to include more of a gaming element like what was done in the NosferARtu project. By making the application more of a game the users will be more engaged with the application. This could also be made into a social experience which was one of the motivating factors that was found in the art gallery example [6].

With the modularity that comes from the implementation using *scriptable objects*, the project can be the base for any number of different applications. For example, if the Living Pa wanted to create an AR application they could use this project as a base and just need to create the 3D models and QR codes.

Overall the project was successful in implementing the AR experience for the Russell Museum and allows for future development to be done on top of the project.

References

[1] H. Ahmad, A. Butt, and A. Muzaffar, "Travel before you actually travel with augmented reality – role of augmented reality in future destination," *Current Issues in Tourism*, vol. 26, no. 17, p. 2845–2862, 2022.

[2] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, "Augmented reality: a class of displays on the reality-virtuality continuum," in *Telemanipulator and Telepresence Technologies*, H. Das, Ed., vol. 2351, International Society for Optics and Photonics. SPIE, 1995, pp. 282 – 292. [Online]. Available: https://doi.org/10.1117/12.197321

[3] "F-35 Gen III helmet mounted display system (HMDS)," Collins Aerospace. [Online]. Available: https://www.collinsaerospace.com/what-we-do/industries/military-and-defense/displays-and-controls/airborne/helmet-mounted-displays/f-35-gen-iii-helmet-mounted-display-system

[4] "HUDWAY Drive: The heads-up display that lets you keep your eyes on the road," HUDWAY. [Online]. Available: https://hudway.co/drive

[5] P. Mésároš, T. Mandičák, A. Mesarosova, M. Hernandez, B. Kršák, C. Sidor, L. Strba, M. Molokáč, L. Hvizdák, P. Blistan, and R. Delina, "Use of augmented reality and gamification techniques in tourism," *e-Review of Tourism Research*, vol. 13, pp. 366–381, 01 2016.

[6] M. C. tom Dieck, T. Jung, and D.-I. Han, "Mapping requirements for the wearable smart glasses augmented reality museum application," *Journal of Hospitality and Tourism Technology*, vol. 7, no. 3, p. 230–253, 2016.

[7] “AR Anchor Manager component,” Unity. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.1/manual/features/anchors.html
[8] “AR Plane Manager component,” Unity. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.1/manual/features/plane-detection.html
[9] “AR Raycast Manager component,” Unity. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.1/manual/features/raycasts.html
[10] “AR Tracked Image Manager component,” Unity. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.1/manual/features/image-tracking.html
[11] “ARCore supported devices,” Google. [Online]. Available: https://developers.google.com/ar/devices
[12] E. Belinski, “Android API Levels.” [Online]. Available: https://apilevels.com/
[13] J. Nielsen, “10 usability heuristics for user interface design,” 1994. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/
[14] “Android Debug Bridge (adb),” Android. [Online]. Available: https://developer.android.com/tools/adb
[15] “Unity’s AR Foundation Framework,” Unity. [Online]. Available: https://unity.com/unity/features/arfoundation
[16] M. Jahn, “ZXing.Net.” [Online]. Available: https://github.com/micjahn/ZXing.Net